Amendments to the Specification:

Please replace paragraph [0008] with the following amended paragraph:

[0008] None of the current approaches to hash-based value numbering apply to machines with instructions that generate more than one result, and which have registers which can store more than one result. The latter are termed "superword registers". For ease of description, we will work with superword registers that can store four 32-bit values; each storage [[are]]area of the superword register is a "component"; we label the four components "x", "y", "z", "w". Note ~~thought~~ though that the method we describe is equally applicable to any combination of superword registers of any size (where "size" is the number of components) and machines with instructions that return any number of results.

Please replace paragraph [0013] with the following amended paragraph:

[0013] Instructions operating on a superword register can also support specialized ~~feature~~ features which further complicate the picture. A first such feature is a swizzling operation, wherein the instruction allows for the re-ordering of the superword values, more specifically the components having associated values. A second possible feature is a write mask which indicates which components an instruction writes in the superword register and which components the superword register utilize previous values.

Please replace paragraph [0014] with the following amended paragraph:

[0014] Current hash based value numbering techniques cannot properly and efficiently process ~~instruction~~ instructions associated with superword registers. Therefore, there exists a

need for a hash based value numbering approach optimizing compiler operations for instructions associated with superword registers.

Please replace paragraph [0020] with the following amended paragraph:

[0020] Briefly, a method and apparatus for superword register value numbering includes hashing an operation code, otherwise referred to as an opcode, and the value numbers of all inputs (i.e., operands) to generate a first hash value. The opcode represents the instruction, for exemplary purposes only, an add, multiply, or any other suitable operation. The value numbers of the inputs are the result value numbers for the definitions of these inputs. The method and apparatus further includes retrieving an operation value number from the first hash table based on the first hash value. The operation value number is an n-tuple value number, wherein n represents the number of components of the superword register.

Please replace paragraphs [0022] through [0025] with the following amended paragraphs:

[0022] More specifically, FIG. 1 illustrates a flowchart of the steps of one method for superword register value numbering. The method begins, step 100, by hashing an operation code and the value numbers of [[to]]the inputs to the instruction to generate a first hash value, step 102. The representation of an instruction includes a previous bit and a write mask, as described in further detail in the U.S. Patent Application~~, having reference number 00100.03.0040 entitled "Method and apparatus for static single assignment form dead code elimination," having the common assignee as the present application~~ entitled "Method and Apparatus for Static Single Assignment Form Dead Code Elimination", having attorney reference number 00100.03.0040, Application No. 10/767,480 and a common assignee. Typically the instruction includes two

operands ~~value numbers~~ and the ~~operand~~ opcode which may be addition, subtraction, multiplication, division, an equivalence operation, for example a equals b, or any other suitable operation.

[0023] Step 104 is retrieving an operation value number from a first hash table based on the first hash value. In one embodiment, a hash table may be pre-loaded with hash values or the hash table may be populated during the compilation process. This step is performed by hashing the combination of the value numbers and the ~~operand~~ opcode using any suitable hashing technique as recognized by one having ordinary skill in the art. Moreover, the step of retrieving is in accordance with known data retrieval techniques, such as any suitable database access routine.

[0024] Step 106 is generating a result value number based on a previous ~~bit hash~~ value number and the operation value number. The result value number is calculated based on the specific components of the superword register~~, such that the~~. The write mask is utilized to determine which components are determined by the operation value number components and which components are determined by a previous value number ~~components~~, retrieved from a previous value hash table. The examination of each component, such as the x component, the y component, the z component and the w component in an exemplary 4 component superword register, generates the result value number.

[0025] Step 108 is searching a second hash table using the result value number. The second hash table may be preloaded with partial data prior to performing the compilation process or may be populated as the compilation process progresses. The second hash table is accessed and searched using standard database access techniques. In this step, [[the]]a determination is made [[if]]as to whether the result value number is within the second hash table, which indicates

if the particular instruction has been previously encountered. Through the utilization of two separate hash tables, the superword instruction may be efficiently subjected to value numbering techniques. Thereupon, in this embodiment the method is complete, step 110.

Please replace paragraphs [0027] through [0030] with the following amended paragraphs:

[0027] If an entry for the first hash value is found within the first hash table, step 126 is retrieving an operation value number from the first hash table. The operation value number is a n-tuple number, where n corresponds to the number of components of the superword register. Therefore, the operation value number contains a value for each of the components. In the event a hash table entry is not located, the operation value number is generated for the first hash value, wherein the new operation value is the n-tuple number ~~assigned the value numbers~~. This operation value number is written to the first hash table.

[0028] Step 128 is retrieving the previous [[bit]]hash value number. This step is performed, in one embodiment, by hashing the previous bit in the instruction~~, that is, the value numbers for the values in the register prior to the current instruction~~. The hashed previous bit is then provided to a previous bit hash table for accessing the table in accordance with known database access techniques and a previous value number is retrieved.

[0029] Step 130 is generating a result value number based on the previous ~~bit hash value~~ number and the operation value number, which is similar to step 106 of FIG. 1. For each component of the superword register, both a value number corresponding to the previous bit, and a value number corresponding to the operation value[[,]] number are on hand. The result value for the component is chosen from the two; if the component is being written to by the current

instruction, the operation value number is selected. Otherwise the previous ~~result~~-value number is selected. The result is an n-vector of value numbers.

[0030] Step 132 is searching a second hash table using the result value number, which is similar to step 108 of FIG. 1. Step 134 is if the result value number is found within the second hash table, retrieving an output of the instruction from the second hash table. The output of the instruction is the resulting operation of the ~~operand~~ opcode and the ~~value numbers~~ operands (including the previous value number). For example, if the ~~operand~~ opcode is addition, the output would be the sum of the value numbers. In this second hash table, the value numbers are mapped to particular instructions. Furthermore, if the result value is found, this indicates that the instruction has been previously encountered and is therefore redundant so that the output of the instruction currently found within the second hash table may be used in place of the current instruction.

Please replace paragraph [0032] with the following amended paragraph:

[0032] FIG. 3 illustrates an apparatus for superword register value numbering. A processor 150 is operatively coupled to a memory 152. The memory stores executable instructions 154 therein. The processor 150 may be, but is not limited to, a single processor, a plurality of processors, a DSP, a microprocessor, an ASIC, a state machine, or any implementation capable of processing and executing software. The term processor should not be construed to refer exclusively to hardware capable of executing software, and may implicitly include DSP hardware, ROM for storing software, RAM, and any other volatile or non-volatile storage medium. The memory with executable instructions 152 may be, but not limited to, a single memory, a plurality of memory locations, shared memory, CD, DVD, ROM, RAM,

EEPROM, optical storage, microcode or any other non-volatile storage capable of storing digital data for use by the processor 150.

Please replace paragraphs [0037] through [0039] with the following amended paragraphs:

[0037] FIG. 4 illustrates a graphical representation of the superword value registering of one embodiment of the present invention. An instruction 170 includes an operation code 172, a first value number 174, a second value number 176, a previous bit 178 and a write mask 180. The operation code 172, the first value number 174 and the second value number 176 are hashed using any suitable hashing technique. The hash value 182 is provided to the first hash table 184. In accordance with standard database access techniques, if the hash value 182 is found, an operation value number 186 is retrieved therefrom. If the hash value 182 is not found within the first hash table 184, the operation value number 186 is generated, as discussed above, written to the first hash table 184 and then retrieved therefrom.

[0038] The previous bit 178 is hashed using any suitable hashing technique and a hashed previous bit 188 is provided to a previous bit hash table 190. Using standard database access techniques, a previous [[bit]]value number 192 is retrieved from the previous bit hash table 190.

[0039] A routine 194 thereupon performs an operation on the operation value number 186, the previous [[bit]]value number 192 and the write mask 180. The routine 194 is based on the per component delineation of the superword register. For each component, a determination is made if the write mask indicated that the value is to effected by the instruction, for example set to a true value. If the component is to be effected, a result value number 196 for that component is the operation value number for that component. If the component is not to be effected, the result value number 196 for that component is the previous value number for the component.

For example, if the instruction is directed to the y component and the z component in a four component superword register, the x and w components would be the previous x and the previous w values and the y and z components would be the operation value number y and z components.

Please replace paragraphs [0041] through [0043] with the following amended paragraphs:

[0041] FIG. 5 illustrates a flowchart of one embodiment of the present invention. Using an exemplary four-word superword register, the first step is defining two exemplary variables r2 and r3 for each of the registers, or components. The steps of FIG. 5 represent programming instructions that may be consumed by a compiler to generate machine language. In step 200, the variable r2 is equivalent to (0.1, 0.2, 0.3, 0.4) and r3 is equivalent to (0.5, 0.6, 0.7, 0.8) for the registers (x, y, z, w). It should also be noted that the underscore "_" indicates a particular register not being written to, otherwise referred to as masked off. Also, previous [[bit]]value numbers are denoted by the symbol "@". It is also noted that for each step, new defined values for the variables r1 through r5 are illustrated in accompanying boxes right of the instruction, where applicable.

[0042] Step 202 is to multiply r2 and r3 to generate the variable r1 for the x[[,]] and y registers because the x and y registers are not masked off (e.g., the write mask value is false for the x and y registers). As noted in step 202, registers z and w are masked off (e.g., the write mask value is true for the z and w registers). Step 204 is adding r3 with r3 with the previous bit r1. As noted in step 204, the addition is specific for register z and w and registers x and y are masked off.

[0043] Step 206 is moving the ~~registers from~~ contents of the register associated with r3 to the ~~registers of~~ register associated with r2. Step 208 is adding r2 with r2 with the previous bits of r1. As noted in step 208, the z and w registers are masked off and the addition is performed for the x and y registers. Step 210 is squaring the registers of r2 to compute registers for r4. As noted, step 210 does not contain any of the four registers masked off. Step 212 is adding r2 with r2 with the previous bits of [[r4]]r1. As noted in step 212, the z and w bits are masked off.